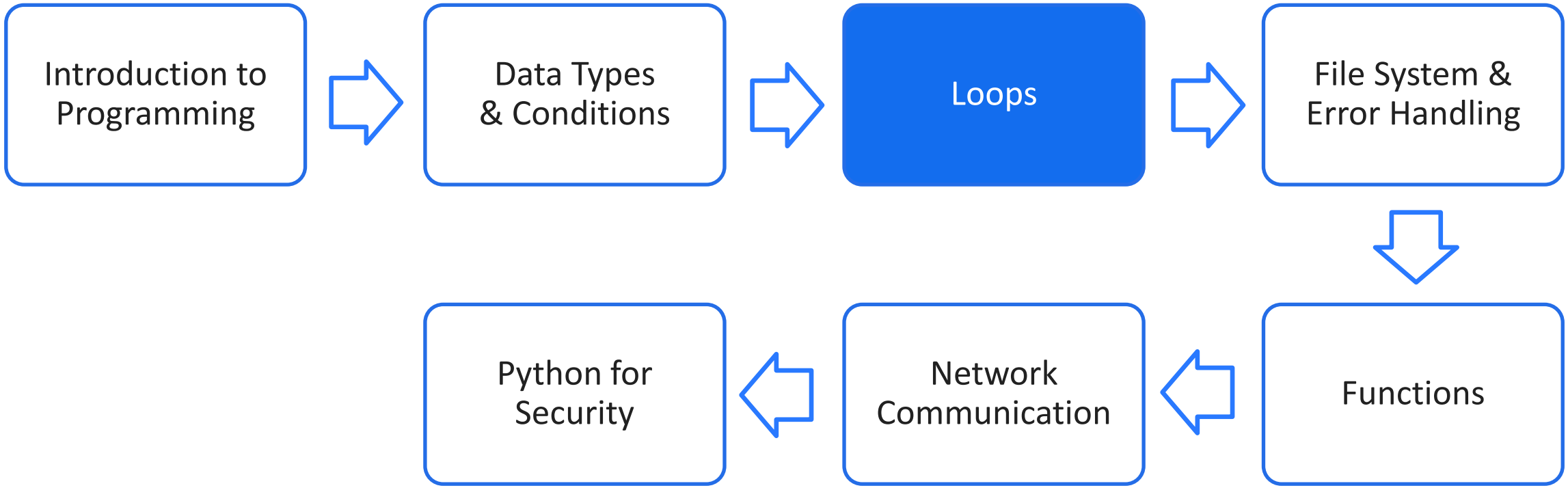
Cybersecurity Professional Program

Loops

Introduction to Python for Security

Course Path


Introduction to Programming
Data Types & Conditions
Loops
File System & Error Handling
Functions
Network Communication
Python for Security

Objectives

This lesson is an introduction to Python ***for*** and ***while*** loops.

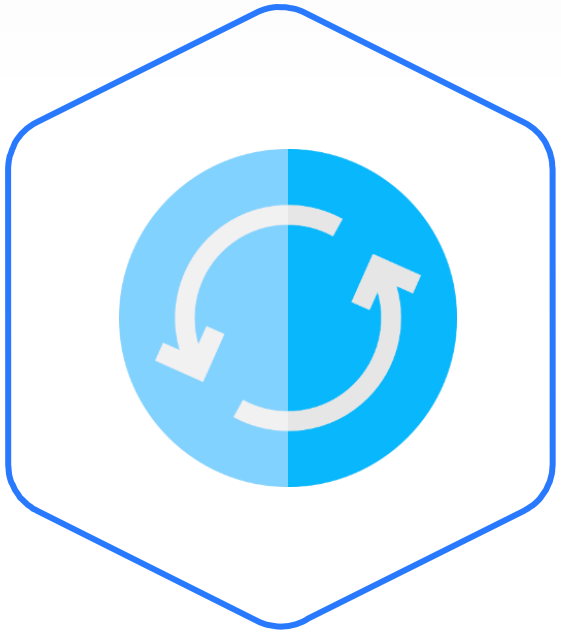
During the lesson, management methods will be pointed out to emphasize organized and efficient code development.

- For & While Loops
- Loops & Conditions

Loops

For & While Loops

What Are Loops?



- Loops are blocks of code that run as long as a condition is true.
- They allow repeating code logic for as long as the condition is true.

For Loops

- Perform blocks of code if the condition is true
- A ***for*** loop provides iterating capabilities.
- Can iterate over lists, defined ranges, and more

```python
word = "test"
num = (1,2,3,4)

for i in word:
    print(i)

print()

for n in num:
    print(n)
```

```
t
e
s
t

1
2
3
4
```

Range vs. List in Loops

- List iteration allows printing data.
- A set range, defined by ***range()***, is also capable of iteration.
- A variable is defined within the loop as the iterating item.

```python
fruit = ["Watermelon", "Banana", "Apple", "Pineapple", "Strawberry"]

for i in fruit:
    print(i, end=" ")

print("\n")

for i in range(0,5):
    print(i, end=" ")

print("\n")

for i in range(0,len(fruit)):
    print(i, end=" ")
```

```
Watermelon Banana Apple Pineapple Strawberry

0 1 2 3 4

0 1 2 3 4
Process finished with exit code 0
```

Calculate Loop Size

The ***len()*** function returns the number of items of an object.

The ***range()*** function accepts an integer and returns the range of the object.

```
namelist = ["David","John","Cooper","Nick"]
print(namelist, "The length is {}".format(len(namelist)))

colorlist = ["Blue","Red","Yellow"]

for i in range(8):
    print(i)
```

```
['David', 'John', 'Cooper', 'Nick'] The length is 4
0
1
2
3
4
5
6
7
```

Lab PY-03-L1

Range Loop

10–20 Min.

Mission

Practice how to create a range with various inputs in a loop.

Steps

- Request a range.
- Iterate the range.
- Display the iterated values.

Environment & Tools

- Python 3
- PyCharm

Related Files

- Lab document

Lab PY-03-L2

Loops in Nested Lists

15–25 Min.

Mission

Practice running nested loops.

Steps

- Open a new project in PyCharm, and create a list called **Classroom**.
- Create a loop for classrooms.
- Create a loop for students.

Environment & Tools

- Python 3
- PyCharm

Related Files

- Lab document

While Loops

- Perform a block of code while a condition is true
- If the condition is false, the ***while*** loop will not be executed.
- ***While*** loops do not have iteration capability like ***for*** loops.

```
counter = 0

while counter <= 6:
    counter += 1
    print(counter)
```

```
1
2
3
4
5
6
7

Process finished with exit code 0
```

Break Command

- Both ***for*** and ***while*** loops can be interrupted.
- The ***break*** command exits a loop.
- Can be placed strategically for flow control

```python
counter = 0

while counter < 50:
    counter += 1
    if counter == 7:
        print("The counter reached the max number: {}".format(counter))
        break
    else:
        print("The current number is: {}".format(counter))
```

```
The current number is: 1
The current number is: 2
The current number is: 3
The current number is: 4
The current number is: 5
The current number is: 6
The counter reached the max number: 7

Process finished with exit code 0
```

Continue Command

- Can be used in both ***for*** and ***while*** loops
- The ***continue*** command skips an iteration.
- Behaves as if the code reached the end of the block

```python
counter = 0

while counter < 10:
    counter += 1

    if counter == 7:
        print("Skipping one increment")
        continue

    print("Current number: {}".format(counter))
```

```
Current number: 1
Current number: 2
Current number: 3
Current number: 4
Current number: 5
Current number: 6
skipping one increment
Current number: 8
Current number: 9
Current number: 10

Process finished with exit code 0
```

Pass Command

- ***Pass*** statements are used as empty executions.
- This is useful for when we don't want to execute any code.
- ***Pass*** can be used as a placeholder for future code.

```
counter = 0

while counter < 50:
    counter += 1
    if counter == 7:
        pass
    else:
        print("The current number is: {}".format(counter))
```

```
The current number is: 1
The current number is: 2
The current number is: 3
The current number is: 4
The current number is: 5
The current number is: 6
The current number is: 8
The current number is: 9
The current number is: 10
The current number is: 11
```

Pass vs. *Continue*

Pass means move on to the remaining code or loop body.

Continue forces the loop to begin the next iteration.

```
word = ("test")
for x in word:
    if x == "s":
        print("Pass execute")
        pass
    print(x)

print()

for x in word:
    if x == "s":
        print("Continue execute")
        continue
    print(x)
```

```
t
e
Pass execute
s
t

t
e
Continue execute
t
```

Loops

Loops & Conditions

Mixing Conditions and Loops

- Conditions have a major role in a loop's flow control.
- They provide greater flexibility for loops.
- They also ensure more efficient conditional checks and help build a better code writing structure.

```
numlist = [10, 22, 44, 30]

for i in numlist:
    if numlist[0] > 6 and numlist[3] > 40:
        print("Both numbers are bigger")
    elif numlist[1] == 22 or numlist[2] == 60:
        print("One of the numbers is equal")
        break
    else:
        print("None of this is true")
```

```
One of the numbers is equal

Process finished with exit code 0
```

Infinite Loops

A loop becomes infinite if a condition is never *false*.

To avoid an infinite loop, the loop must verify the existence of a false condition.

```
num1 = 1
while num1 == 1:
    user = input("Please enter a number: ")
    print("You have entered the number {}".format(user))

print("Good Bye!")
```

```
You have entered the number 11
Please enter a number: 44
You have entered the number 44
Please enter a number: 66
You have entered the number 66
Please enter a number: 88
You have entered the number 88
Please enter a number: 46
You have entered the number 46
```

Nested Loops and Lists

The ***append()*** function can be used to define a list within another list.

append() adds a value to an existing variable.

Nested lists can be used to arrange data in hierarchical structures.

```
numlist = []

for i in range(5):
    numlist.append([])
    for b in range(5):
        numlist[i].append(b)
print(numlist)
```

```
[[0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]

Process finished with exit code 0
```

Lab PY-03-L3

Loops with Conditions

10–20 Min.

Mission

Practice executing code with a ***while*** loop and conditions.

Steps

- Create a counting mechanism.
- Locate a specific value when counting.

Environment & Tools

- Python 3
- PyCharm

Related Files

- Lab document

Lab PY-03-L4

User Dictionary

15–20 Min.

Mission

Work with input from users to control program flow and insert and retrieve the desired output.

Steps

- Create an empty dictionary and two variables.
- Identify the service.
- Print the final dictionary if the user breaks the loop.

Environment & Tools

- Python 3
- PyCharm

Related Files

- Lab document

Lab PY-03-L5

While and Conditions

30–50 Min.

Mission

Implement conditions to control infinite ***while*** loops.

Steps

- Create a list of groceries with an available budget.
- Iterate the list for budget calculation.
- Simulate grocery shopping.

Environment & Tools

- Python 3
- PyCharm

Related Files

- Lab document

Thank You

Questions?